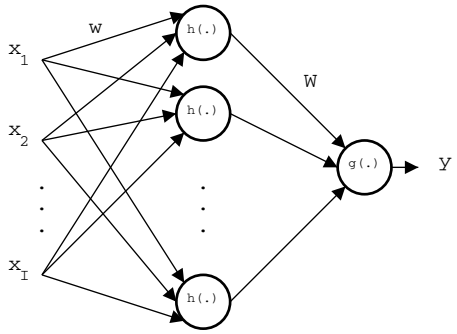
1 Meglio prevenire che curare

In qualità di consultenti presso una nota ditta automobilistica ci è stato chiesto di realizzare un sistema automatico per la diagnosi preventiva dei loro motori. Tale sistema deve essere in grado di classificare un motore come "BUONO" o "NON_BUONO" durante il suo funzionamento sulla base di misurazioni meccaniche, analisi chimiche dei gas di scarico e rumore di funzionamento.

Affascinati da questa nuova tecnologia chiamata **Reti Neurali**, decidiamo di utilizzarla nello sviluppo del sistema in questione e in particolare facciamo riferimento alla seguente figura.



1.1 I dati a disposizione [Punti 2]

Il nostro committente non è in grado di darci chiare indicazioni sui parametri caratteristici di un motore buono o meno, quello che può fornirci è semplicemente un archivio di misurazioni prese sulle macchine portate nelle loro officine autorizzate in corrispondenza dei tagliandi periodici e del giudizio sul motore fatto da un tecnico specializzato in quei casi. La cosa ci preoccupa? Perchè?

La cosa non ci preoccupa perchè il problema si configura come un tipico caso di apprendimento supervisionato. Il compito della rete neurale sarà apprendere la funzione ignota che lega le quantità misurate al giudizio del tecnico sul motore direttamente a partire dai dati. Qualora avessimo già avuto indicazioni sui parametri caratteristici di motori buoni o meno, avremmo probabilmente utilizzato un sistema esperto e non una rete neurale.

1.2 La scelta del modello (Parte 1) [Punti 2]

Il primo problema che ci si presenta nella realizzazione del sistema è la scelta dell'architettura della rete neurale e qui la cosa è piuttosto semplice, tra un singolo percettrone e un percettrone multistrato scegliamo il secondo. Si, ma perchè?

Il percettrone singolo possiede minime capacità di discriminazione; in uno spazio a N dimensioni la superfice di separazione è un ierpiano. Nel nostro caso non sappiamo a priori quale sarà la forma della superfice di sepaazione tra le due

classi, ma difficilmente sarà un semplice iperpiano. Scegliamo quindi di usare un percettrone multistrato dato che riesce ad approssimare superfici di separazione complesse e, all'occorrenza, anche iperpiani.

1.3 Funzioni di attivazione [Punti 2]

Perfetto! Abbiamo la nostra bella architettura multistrato; diciamo che ci basta un solo strato nascosto per questione di semplicità e leggiamo un pò in giro per capire che funzioni di attivazione mettere nello strato in questione. Quasi tutti, citando un articolo che non abbiamo voglia di andare a cercare, dicono che la scelta giusta è la sigmoide e noi decidiamo di fare altrettanto, ma leggiamo anche che la funzione di attivazione dello strato di uscita dipende dalla codifica scelta per il particolare problema.

Nel nostro caso, anche sforzandosi, non ci sono molte possibilità abbiamo una sola uscita con due soli valori "BUONO" o "NON_BUONO" che scelgliamo di rappresentare con i valori "1" e "0" per comodità implementativa del circuito a valle del classificatore neurale. Posto questo vincolo quale sarà la funzione di attivazione dello strato di uscita e perchè?

La funzione di attivazione dovrà essere limitata tra 0 *e* 1 *per compatibilità con il resto del sistema. Inoltre lavorando con un percettrone multistrato è necessario che sia derivabile per permettere un addestramento tramite back-propagation. La scelta ricade quindi sulla funzione sigmoide anche per lo strato di uscita in quano limitata all'intervallo* (0 1) *e derivabile.*

1.4 La funzione d'errore [Punti 3]

Un tempo lavoravamo per conto di una società che faceva analisi dei dati e sappiamo benissimo che in caso di variabili binarie la distrbuzione di probabilità che descrive meglio il fenomeno osservato è una Bernulliana $t_n \sim Be(\theta)$ con parametro caratteristico θ pari alla probabilità che si verifichi l'evento "1". La probabilità di un'osservazione in questo caso quindi è $P(t_n) = \theta^{t_n}(1-\theta)^{(1-t_n)}$.

Cerchiamo di usare questa informazione a nostro vantaggio e decidiamo di usare la nostra rete neurale come modello non lineare che, dalle osservazioni sul campo, cerca di prevedere la probabilità che si verifichi l'evento t_n. Questo vuol dire in pratica che l'uscita della mia rete y dovrà approssimare la probabilità θ che descrive il mio fenomeno e che quindi la probabilità di un particolare evento diventa $P(t_n) = y^{t_n}(1-y)^{(1-t_n)}$. Usando una stima a massima verosomiglianza questo mi riporta al semplice problema di minimizzare la seguente funzione d'errore:

$$-\sum_{n}^{N} t_n \log y_n + (1-t_n)\log(1-y_n)$$

Quali sono i passaggi matematici che portano dalla probabilità del singolo evento a tale funzione d'errore?

Per usare una stima a massima verosomiglianza bisogna calcolarsi come prima cosa la probabilità del campione osservato in funzione dei parametri del modello

(uscita y della rete neurale). Nel nostro caso per l'indipendenza del campione osservato abbiamo:

$$P(t_1, t_2, \ldots, t_N) = P(t_1) \cdot P(t_2) \cdot \ldots \cdot P(t_N) = \prod_n^N P(t_n)$$

dato che la probabilità del singolo campione è data da $P(t_n) = y^{t_n}(1-y)^{(1-t_n)}$ *otteniamo per la verosomiglianza:*

$$L = \prod_n^N y^{t_n}(1-y)^{(1-t_n)}.$$

Ricordando che a noi interessa il massimo di tale funzione e che la funzione logaritmo, essendo monotona, preserva il massimo ottenimo la seguente funzione da massimizzare:

$$\begin{aligned} l = \log L &= \log\left(\prod_n^N y^{t_n}(1-y)^{(1-t_n)}\right) \\ &= \sum_n^N \left(\log y^{t_n}(1-y)^{(1-t_n)}\right) \\ &= \sum_n^N \left(\log y^{t_n} + \log(1-y)^{(1-t_n)}\right) \\ &= \sum_n^N \left(t_n \log y + (1-t_n)\log(1-y)\right) \end{aligned}$$

Massimizzare la veromisglianza equivale quindi a minimizzare l'opposto della precedente funzione $-l = -\sum_n^N \left(t_n \log y + (1-t_n)\log(1-y)\right)$.

1.5 L'algoritmo di apprendimento [Punti 3]

Cosa ci resta da fare? Ah si, dobbiamo trovarci i pesi che minimizzano questo errore. Basta applicare la regola di back-propagation anche nota come discesa del gradiente. Calcoliamoci quindi il gradiente del nostro errore rispetto ai pesi W_j e w_{ji} della nostra rete.

Ricordiamo che, come da disegno di riferimento, l'uscita di una rete neurale con I ingressi, J neuroni nello strato nascosto e una sola uscita è:

$$y = g(\sum_j^J W_j h(\sum_i^I w_{ij} x_i)).$$

Posto $a_j = \sum_i^I w_{ij} x_i$, $b_j = h(a_j)$ *e* $A = \sum_j^J W_j b_j$ *Calcoliamo le derivate della funzione da minimizzare rispetto a* W_j *e* w_{ij}.

$$
\begin{aligned}
\frac{\partial E}{\partial W_j} &= -\sum_n^N \left[\frac{\partial (t_n \log y)}{\partial W_j} + \frac{\partial ((1-t_n)\log(1-y))}{\partial W_j} \right] \\
&= -\sum_n^N \left[t_n \frac{1}{y} \frac{\partial y}{\partial W_j} + (1-t_n) \frac{1}{(1-y)} \frac{\partial (-y)}{\partial W_j} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) \frac{\partial A}{\partial W_j} - \frac{1-t_n}{1-y} g'(A) \frac{\partial A}{\partial W_j} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) b_j - \frac{1-t_n}{1-y} g'(A) b_j \right],
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ij}} &= -\sum_n^N \left[\frac{\partial (t_n \log y)}{\partial w_{ij}} + \frac{\partial ((1-t_n)\log(1-y))}{\partial w_{ij}} \right] \\
&= -\sum_n^N \left[t_n \frac{1}{y} \frac{\partial y}{\partial w_{ij}} + (1-t_n) \frac{1}{(1-y)} \frac{\partial (-y)}{\partial w_{ij}} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) \frac{\partial A}{\partial w_{ij}} - \frac{1-t_n}{1-y} g'(A) \frac{\partial A}{\partial w_{ij}} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) W_j \frac{\partial h(a_j)}{\partial w_{ij}} - \frac{1-t_n}{1-y} g'(A) W_j \frac{\partial h(a_j)}{\partial w_{ij}} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) W_j h'(a_j) \frac{\partial a_j}{\partial w_{ij}} - \frac{1-t_n}{1-y} g'(A) W_j h'(a_j) \frac{\partial a_j}{\partial w_{ij}} \right] \\
&= -\sum_n^N \left[\frac{t_n}{y} g'(A) W_j h'(a_j) x_i - \frac{1-t_n}{1-y} g'(A) W_j h'(a_j) x_i \right].
\end{aligned}
$$

1.6 Attenzione all' overfitting [Punti 2]

Ben fatto, l'esame di analisi è servito a qualcosa! Adesso mi basta lanciare il mio programma di minimizzazione sul mio corposo dataset e ottenere il mio classificatore neurale. Ma come faccio a evitare minimi locali? E come faccio a evitare il temutissimo fenomeno dell'overfitting?

Il problema dei minimi locali non è completamente risolvibile, un modo per migliorare le performance è far ripartire l'algoritmo di ottimizzazione più volte e scegliendo poi il risultato migliore. Anche l'uso di metodi più raffinati per la minimizzazione di funzioni potrebbe aiutare in questo senso (e.g., discesa del gradiente con momento, gradiente coniugato, etc.).

Il problema dell'overfitting è dovuto principalmente a una sovraparametrizzazione del modello. Una tecnica per ovviare al problema è l'early stopping: uso

parte dei dati di addestramento per stimare l'errore di generalizzazione anzichè usarli come dati per la minimizzazione dell'errore e fermo l'apprendimento quando l'errore di generalizzazione non decresce più. Un'altra tecnica per ridurre l'overfitting è nota con il nome di weight decay: in questo caso la funzione da minimizzare include anche un termine di penalizzazione per pesi sinaptici troppo grandi dovuto a considerazioni sia empiriche sia statistiche.

1.7 La scelta del modello (Parte 2) [Punti 2]

A questo punto ci manca un solo dettaglio scegliere il numero di neuroni nello strto nascosto ... avete qualche idea degna di lode?

Il dimensionamento di una rete neurale è un problema ancora aperto, una tecnica comunemente usata nel caso di percettroni multistrato è, ancora una volta, procedere per tentativi. Si comincia con un numero minimo di neuroni (al limite anche uno solo) e si addestrano diverse reti con numero di neuroni nello strato nascosto sempre crescente. Osservando l'andamento d'errore sul dataset di validazione si noterà che questo decresce fino a un punto per cui aumentare la complessità del modello non comporta significativi miglioramenti ... quello è un buon punto dove fermarsi.